INF8775 – Analyse et conception d’algorithmes

TP2 – Hiver 2023

|  |  |
| --- | --- |
| **Nom, prénom, matricule des membres** | Dépelteau, Nicolas, 2083544  Turcotte, Alexandre, 2087684 |
| **Note finale / 13** |  |

# Informations techniques

* Répondez directement dans ce document. Veuillez ne pas inclure le texte en italique servant de directive.
* La correction se fait sur ce même rapport.

Vous devez faire une remise électronique sur Moodle avant le

14 mars à 23h59 pour le groupe B2

21 mars à 23h59 pour le groupe B1

en suivant les instructions suivantes :

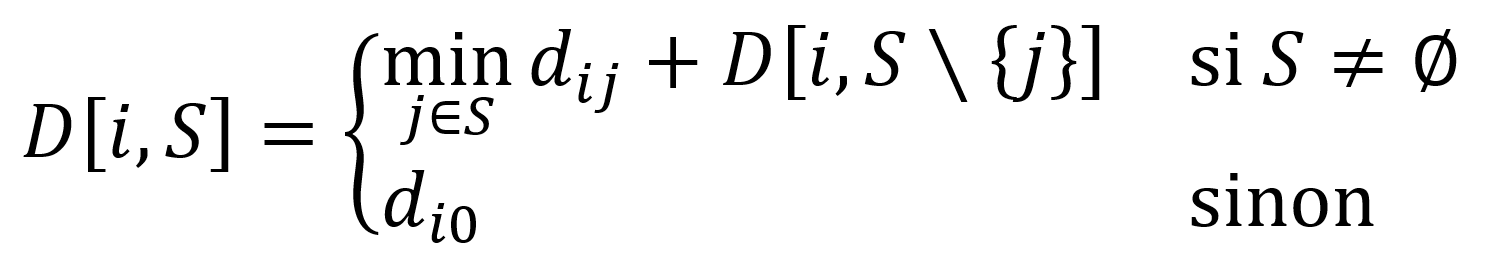
* Vos fichiers doivent être remis dans une archive zip à la racine de laquelle on retrouve :
  + Ce rapport sous format ODT ou docx.
  + Un script nommé *tp.sh* servant à exécuter les différents algorithmes du TP. L’interface du script est décrite à la fin du rapport.
  + Le code source et les exécutables.
* Vous avez le choix du langage de programmation utilisé mais vous devrez utiliser les mêmes langage, compilateur et ordinateur pour toutes vos implantations. **Le code et les exécutables soumis devront être compatibles avec les ordinateurs de la salle L-4714.**
* Si vous utilisez des extraits de codes (programmes) trouvés sur Internet, vous devez en mentionner la source, sinon vous serez sanctionnés pour plagiat.

# Mise en situation

Ce travail pratique se répartit sur deux séances de laboratoire et porte sur l'analyse et la conception d'algorithmes développés suivant différents patrons de conception.

On vous demande d’y résoudre le problème du voyageur de commerce (*Traveling Salesman Problem*, TSP) en utilisant trois patrons de conception distincts :

1. Algorithme glouton : en démarrant d’une ville arbitraire et en faisant une tournée en sélectionnant à chaque fois le plus proche voisin ;
2. Algorithme de programmation dynamique : en résolvant le problème à travers la relation de récurrence suivante :



1. Algorithme approximatif (1-relatif) : en construisant un arbre sous-tendant minimum (Minimum Spanning Tree, MST) du graphe et en le parcourant en préordre à partir d’une ville arbitraire.

On s’intéresse dans ce TP au problème du TSP métrique : les villes appartiennent à un plan N2 muni de la distance euclidienne **arrondie à l’entier le plus proche**.

# Jeux de données

Pour tester les algorithmes, vous devez générer des jeux de données en utilisant le script fourni :

$ ./inst\_gen.py [-h] -s NB\_VILLES [-n NB\_EXEMPLAIRES] [-x PRÉFIXE]

On suggère d’utiliser le script bash suivant pour automatiser la génération des exemplaires :

|  |
| --- |
| #!/bin/bash  # Pour glouton et approx  for n in {"1000","5000","10000","50000","100000"}; do  ./inst\_gen.py -s $n -n 5  done  # Pour tous les algorithmes  for n in {"5","10","15","20","25"}; do  ./inst\_gen.py -s $n -n 5 -x DP  done |

La première ligne de chaque exemplaire contient le nombre de villes à visiter (qui est aussi la taille du problème *N*). Les *N* lignes subséquentes contiennent chacune les coordonnées d’une ville à la fois, séparées par des espaces.

Les coordonnées des villes sont chacune comprises entre 0 et 2000 et aucune paire de villes n’est telle que la distance entre elles est de 0.

# Présentation des résultats

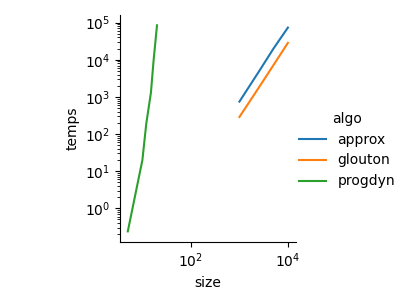
|  |  |
| --- | --- |
| 0 | / 4 pt |

### Tableau des résulats

|  |  |  |
| --- | --- | --- |
| **Algorithme** | **Taille** | **Temps moyen** |
| approx | 1000 | 746,2160 |
| approx | 2500 | 4736,5785 |
| approx | 5000 | 19688,2143 |
| approx | 7500 | 42699,7313 |
| approx | 10000 | 73679,0775 |
| glouton | 1000 | 288,2101 |
| glouton | 2500 | 1765,2129 |
| glouton | 5000 | 7079,6046 |
| glouton | 7500 | 16033,2270 |
| glouton | 10000 | 28572,5018 |
| progdyn | 5 | 0,2386 |
| progdyn | 10 | 19,8563 |
| progdyn | 12 | 186,0145 |
| progdyn | 15 | 1313,4386 |
| progdyn | 17 | 9382,3962 |
| progdyn | 20 | 84777,7278 |

### Graphiques pour analyse hybride

Test de puissance :



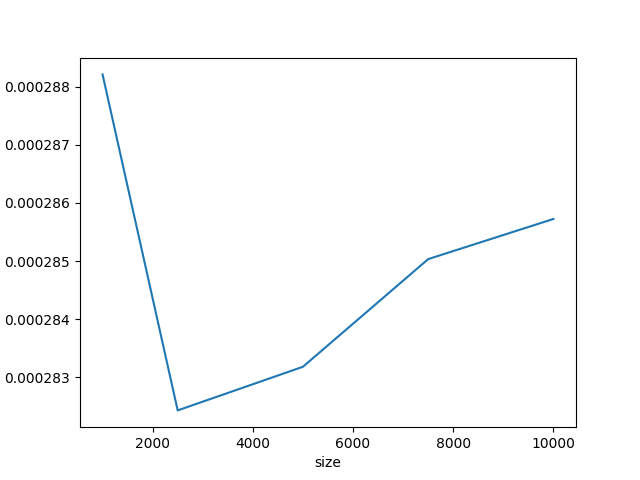
Temps = 2.0015 \* Taille + -7.1982 pour l'algorithme approx

Temps = 1.9970 \* Taille + -8.1384 pour l'algorithme glouton

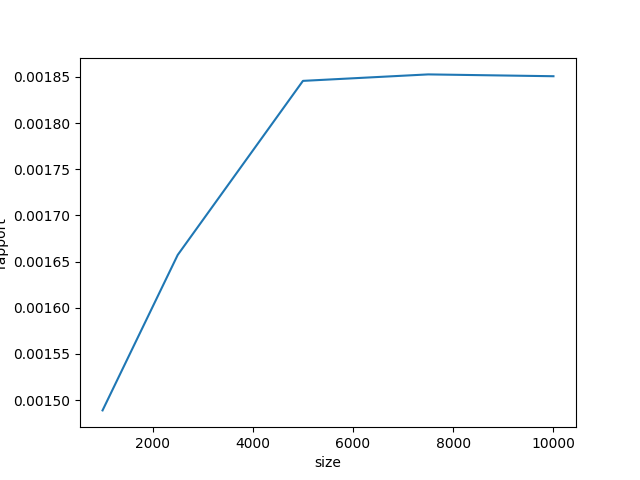
Temps = 9.0562 \* Taille + -16.7981 pour l'algorithme progdyn

Test du rapport :

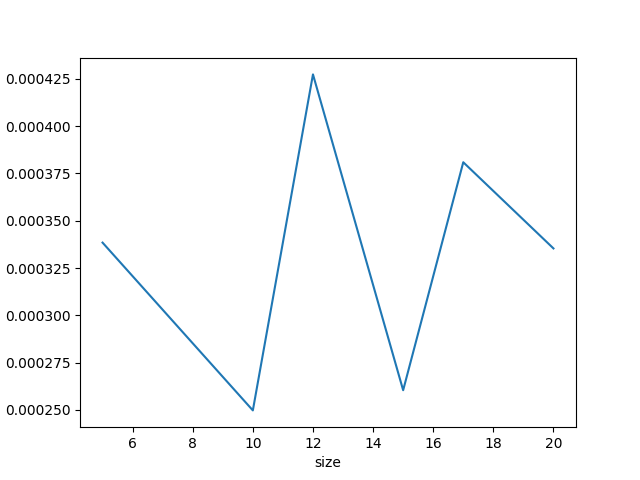
Glouton où f(n) = n2



Approximatif où f(n) = n1.9



Programmation dynamique où f(n) = (1.95)n \* n2



### Analyse et discussion

|  |  |
| --- | --- |
| 0 | / 6 pt |

### Faites une analyse asymptotique du temps de calcul pour chaque algorithme.

Le nom du fichier est anal\_complex\_algo.pdf

### Servez-vous de vos temps d'exécution pour confirmer et/ou préciser l'analyse asymptotique théorique de vos algorithmes avec la méthode hybride de votre choix.

Nous avions estimé l’algorithme glouton avec une complexité de n2 et grâce au graphique du test du rapport, nous pouvons confirmer que notre hypothèse était bonne, car le graphique converge vers une valeur.

Nous avions estimé l’algorithme approximatif avec une complexité de n2, en revanche notre graphique du test du rapport nous dit que notre droite converge à une valeur de n1.9. Puisque cet écart est très petit et le fait que nous avons un petit échantillon, nous pouvons estimer que notre hypothèse est très proche du résultat empirique.

Nous avions estimé l’algorithme de programmation dynamique avec une complexité de (2n)\*n2 et grâce au graphique du test du rapport, nous avons pu estimer notre complexité empirique à (1.95n)\*n2. Encore une fois, la différence entre notre complexité théorique et empirique est très petite et basé sur un petit échantillon, ce qui nous permet d’affirmer que notre hypothèse est très proche du résultat empirique.

### Discutez des trois algorithmes en fonction de la qualité respective des solutions obtenues, de la consommation de ressources (temps de calcul, espace mémoire) et de la difficulté d'implantation.

Pour l’algorithme glouton, nous avons un algorithme plutôt rapide qui demande très peu de mémoire. De plus, il est assez facile à implémenter en quelque ligne.

Pour l’algorithme de programmation dynamique, il demande énormément de temps (même avec des exemplaires de petite taille) et demande aussi beaucoup de mémoire. Il est assez difficile à implémenter et demande d’être inventif quand aux différents choix de structure de donnée et lorsque nous devions manipuler les données.

Pour ce qui est de l’algorithme approximatif, il demande énormément de mémoire avec un temps d’exécution moyen. C’est un algorithme plutôt simple à implémenter à condition d’être à l’aise avec les graphes.

### Indiquez sous quelles conditions vous utiliseriez chaque algorithme.

Nous utiliserions l’algorithme glouton si nous avons des contraintes de temps et de mémoire strictes à respecter.

Nous utiliserions la programmation dynamique dans l’unique cas où nous aurions besoin d’avoir le résultat exact.

Nous utiliserions l’algorithme approximatif dans un cas milieu. C’est-à-dire lorsque nous avons des contrainte de temps et de mémoire assez lousse, mais que nous devons nous assurer d’avoir un résultat proche de la solution optimale.

# On vous fournit 5 exemplaires difficiles à résoudre jusqu’à optimalité[[1]](#footnote-1) (fichiers avec préfixe hard). Tentez de les résoudre à l’aide de vos algorithmes glouton et approximatif et discuter des écarts obtenus.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fichier | Nombre de villes | Solution optimale (L2 arrondie) | Résultat avec Glouton | Résultat avec approximatif |
| hard\_N52 | 52 | 551609 | 632298 | 625836 |
| hard\_N91 | 91 | 1228726 | 1338013 | 1761654 |
| hard\_N130 | 130 | 1928734 | 2125138 | 2353228 |
| hard\_N169 | 169 | 2600546 | 2657248 | 3398780 |
| hard\_N199 | 199 | 3139778 | 3342750 | 3857871 |

L’algorithme glouton obtient des résultats qui sont plus proche de la solution optimale que l’algorithme approximatif. Malgré tout, l’algorithme approximatif respecte sont contrat de fournir une solution meilleure que deux fois la solution optimale.

# Autres critères de correction

## Respect de l’interface tp.sh

|  |  |
| --- | --- |
|  | / 1 pt |

Utilisation :

$ ./tp.sh -a {glouton, progdyn, approx} -e CHEMIN\_EXEMPLAIRE [-p] [-t]

Arguments optionnels :

-p affiche dans l’ordre, sur chaque ligne, les indices des villes à visiter en commençant par 0 et en finissant par 0, sans texte superflu. Rapportez le chemin tel que la deuxième ville visitée ait un indice inférieur à celui de l’avant dernière ville affichée (voir exemple).

-t affiche le temps d’exécution en millisecondes, sans unité ni texte superflu.

Important : l’option -e doit pouvoir accepter des chemins absolus.

user@host folder $ ./tp.sh -p -e N5\_0 -a progdyn

0

**1**

4

3

**2**

0

## Qualité du code

|  |  |
| --- | --- |
|  | / 1 pt |

## Présentation générale

|  |  |
| --- | --- |
|  | / 1 pt |

* Concision
* Qualité du français

## Pénalité retard

|  |
| --- |
| 0 |

* -1 pt / journée de retard, arrondi vers le haut. Les TPs ne sont plus acceptés après 3 jours.

1. Tirés de <http://www.or.uni-bonn.de/~hougardy/HardTSPInstances.html>, les coordonnées peuvent dépasser 2000. [↑](#footnote-ref-1)